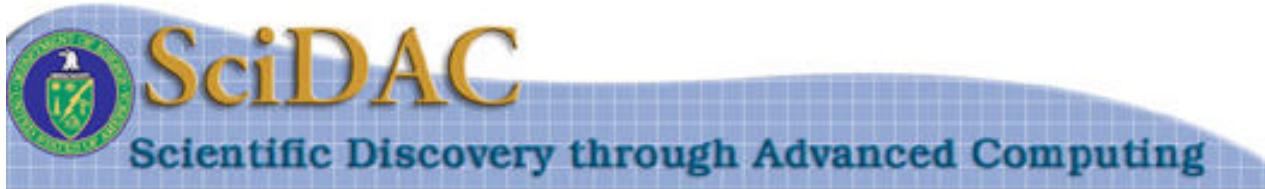


# **PERC Performance Tools: Accomplishments and Vision**

**The PERC Team  
Bronis R. de Supinski  
[bronis@llnl.gov](mailto:bronis@llnl.gov)**

**Center for Applied Scientific Computing  
Lawrence Livermore National Laboratory  
May 5, 2003**

**[http://perc.nersc.gov/tools/tm\\_integration.htm](http://perc.nersc.gov/tools/tm_integration.htm)**



# Objectives

---

## ✍ PERC mission

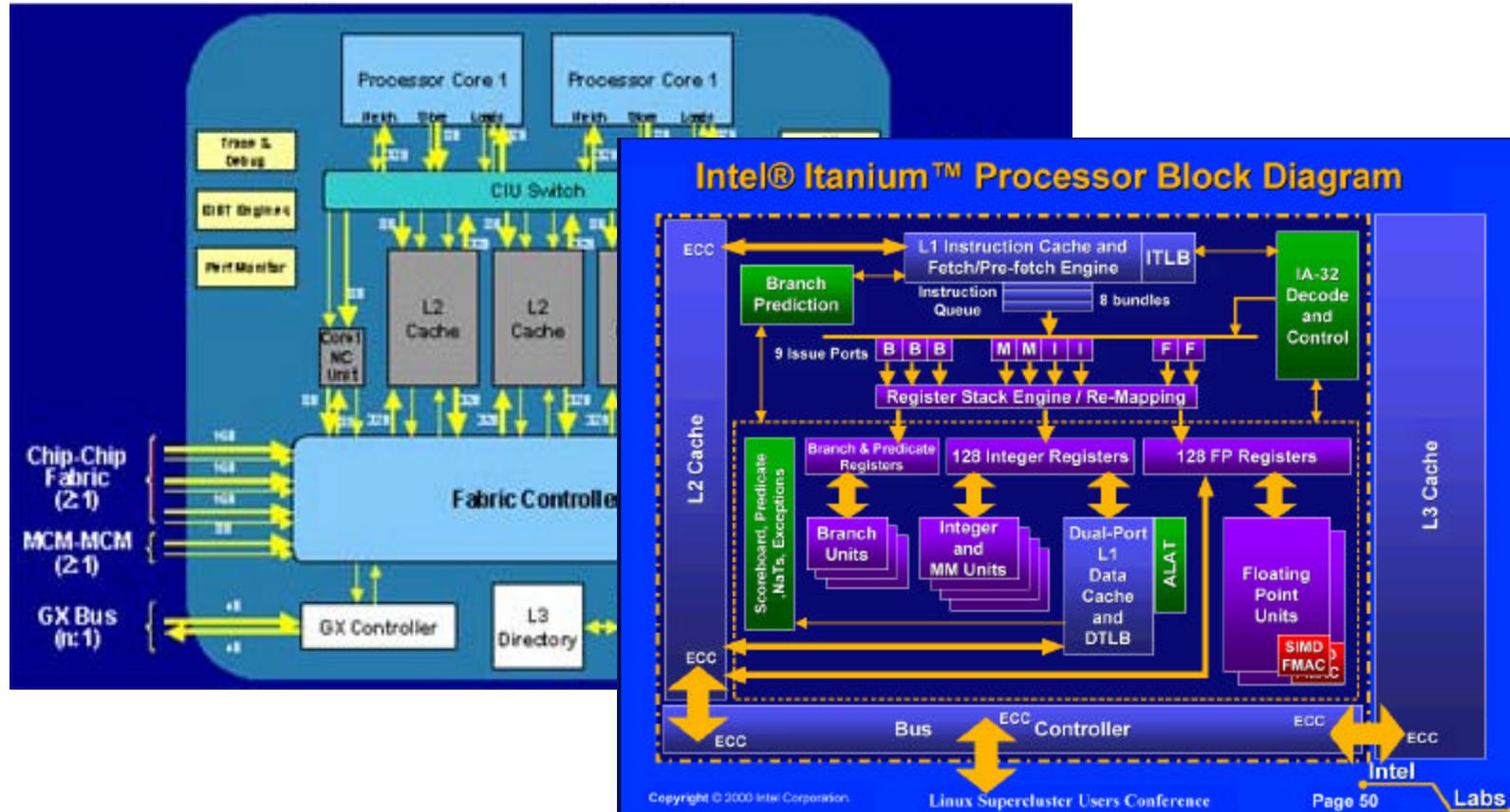
- ✍ develop a science of performance
- ✍ engineer tools for performance analysis and optimization

## ✍ PERC objectives

- ✍ understand key factors affecting performance
  - ✍ in applications and in computer systems
- ✍ develop models to predict application performance
- ✍ *develop an enabling infrastructure of tools*
  - ✍ *performance monitoring, modeling and optimization*
- ✍ validate these ideas and infrastructure
  - ✍ close collaboration with DOE Office of Science and others
- ✍ transfer the technology to end users



# Rising Processor Complexity



- ☛ PERC provides the
  - ☛ tools, models, benchmarks, and insights
- ☛ Needed to understand and optimize
  - ☛ new computer systems and applications



# PERC Performance Tools

---

## Infrastructure

- ☛ PAPI (Tennessee): <http://icl.cs.utk.edu/projects/papi>
- ☛ Dyninst (Maryland): <http://www.dyninst.org>
- ☛ ROSE (LLNL)

## Measurement tools

- ☛ SvPablo (Illinois): <http://www-pablo.cs.uiuc.edu/Software/SvPablo>
- ☛ SIGMA (Maryland)
- ☛ Performeter (Tennessee)
- ☛ dsd Regularity Measurement (LLNL)
- ☛ Dynaprof (Tennessee)



# PERC Performance Tools

---

## ✍ Modeling tools

- ✍ MetaSim (San Diego): <http://www.sdsc.edu/PMaC>
- ✍ Performance Bounding Tool (ANL)

## ✍ Optimization/Tuning tools

- ✍ Performance Assertions (LLNL)
- ✍ ROSE (LLNL)
- ✍ Active Harmony (Maryland): <http://www.dyninst.org/harmony>
- ✍ ATLAS/AEOS (Tennessee)



# Dyninst Memory Instrumentation Features

---

- ✍ **Finding memory access instructions**

- ✍ loads, stores, prefetches

- ✍ **Builds on arbitrary instrumentation**

- ✍ **Decoded instruction information**

- ✍ type of instruction
  - ✍ constants and registers for computing
    - ✍ the effective address
    - ✍ the number of bytes moved
  - ✍ available in the mutator before execution

- ✍ **Memory access snippets**

- ✍ effective address in process space
  - ✍ byte count
  - ✍ available in mutatee at execution time



# Dyninst Memory Instrumentation

---

## ✍ Dynamic memory access instrumentation

- ✍ collect low level memory accesses
  - ✍ flexibility of dynamic instrumentation

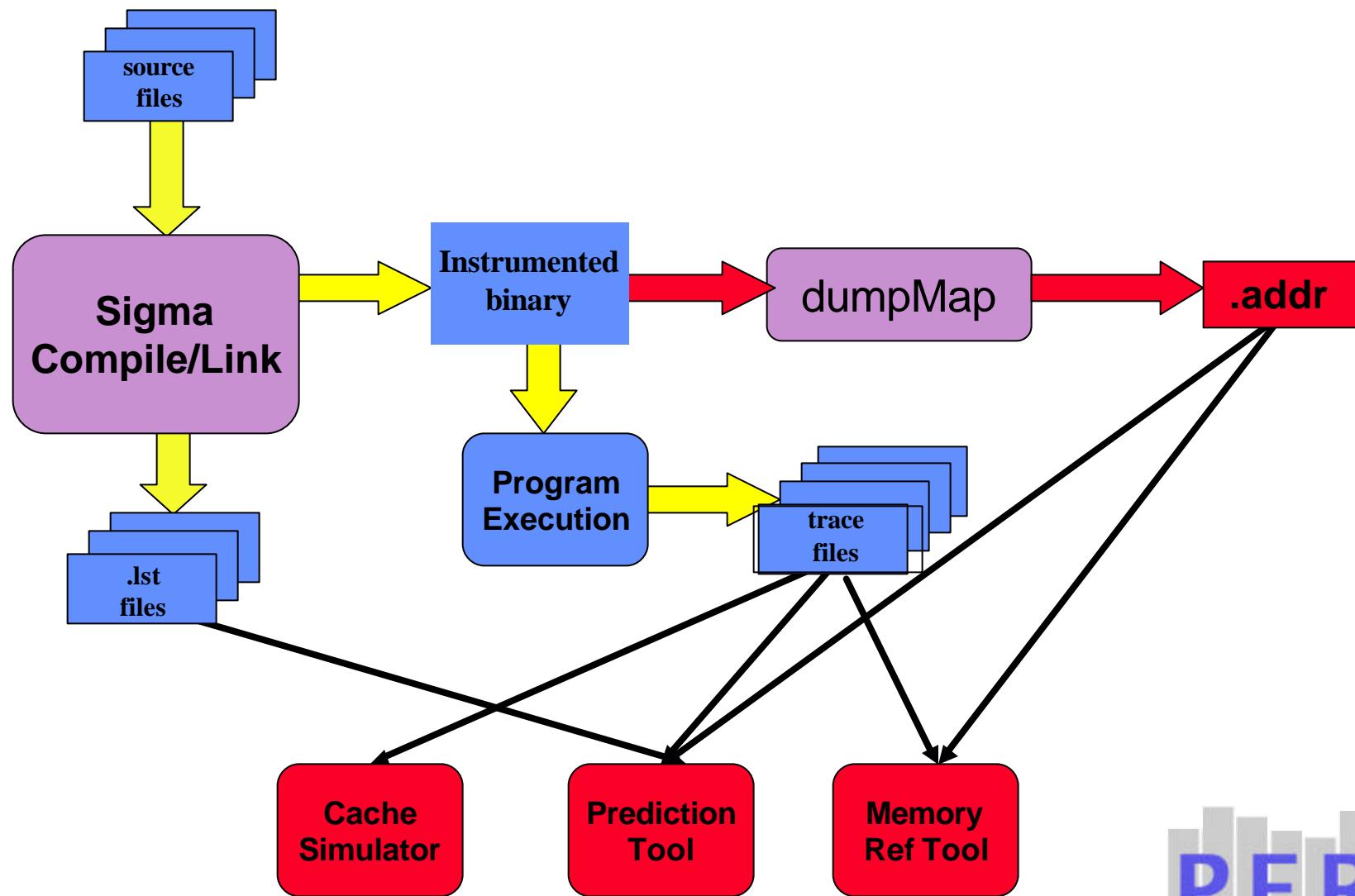
## ✍ Applications

- ✍ offline performance analysis (Sigma re-implementation)
- ✍ online optimization and tools to catch memory errors

## ✍ Sigma using Dyninst (memory/cache understanding)

- ✍ runtime selection of memory to instrument
  - ✍ focus on key procedures
  - ✍ only instrument selected time steps
- ✍ multi-platform support (Sigma only ran on Power3)
- ✍ instrumentation without re-compiling
- ✍ Multiple tools
  - ✍ memory profiler, cache prediction and detailed simulation

# Structure of SIGMA Data Collection



# PAPI Features

---

## ✍ **Performance Application Programming Interface**

- ✍ hardware performance monitor counters
- ✍ PAPI preset events
  - ✍ mapping from symbolic names to machine specifics
  - ✍ cycle counts, cache misses, floating point operations
  - ✍ multiplexing and event management

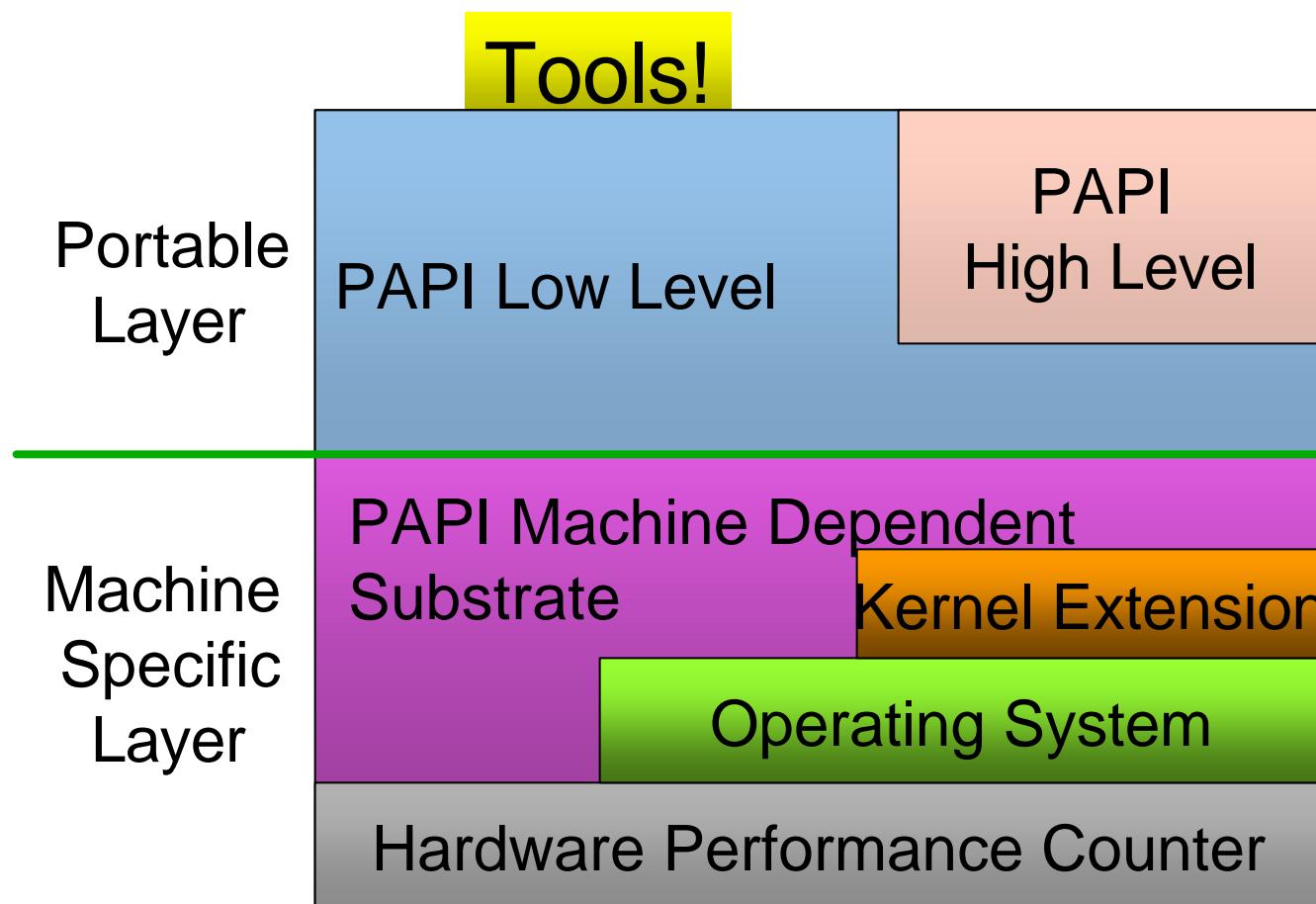
## ✍ **High and low level APIs**

- ✍ high-level provides ease of use with presets
- ✍ low-level provides detailed access and management

## ✍ **Tool uses**

- ✍ SvPablo, HPCView, TAU, ...

# PAPI Implementation



# PAPI Releases

---

## ☛ PAPI 2.3.3 release

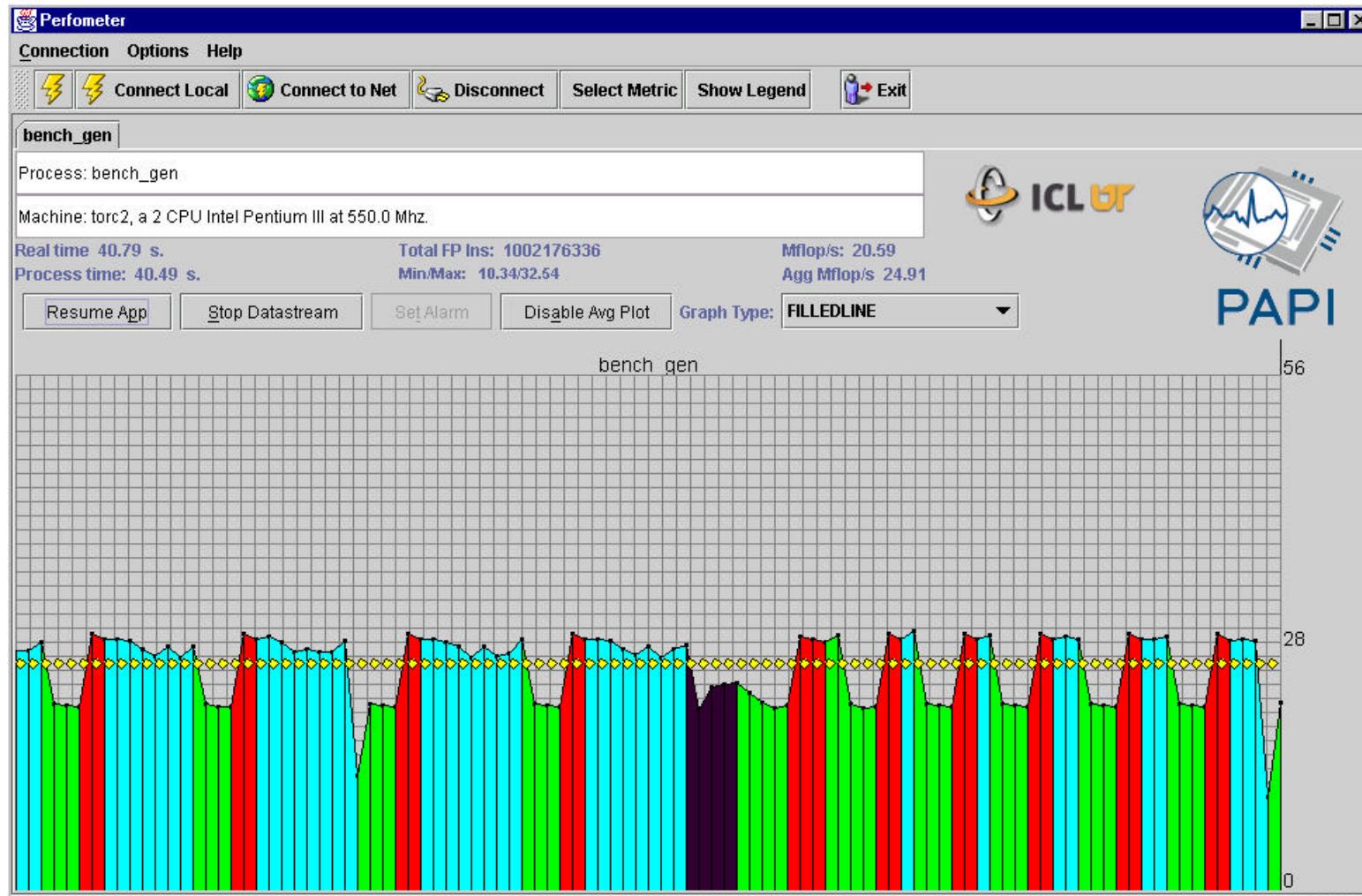
- ☛ additional platforms
  - ☛ IBM PPC604, 604e, Power3, Power4
  - ☛ Intel x86, Sun UltraSparc I/II/III
  - ☛ SGI MIPS R10K/R12K/R14K
  - ☛ Compaq Alpha 21164/21264
  - ☛ Itanium and Itanium2
- ☛ enhancements
  - ☛ static/dynamic memory information and IA-64 hardware profiling
- ☛ sample tools
  - ☛ Perfometer, Trapper and Dynaprof

## ☛ PAPI 3.0 release (scheduled June 2003)

- ☛ redesign for robustness, feature set, simplicity and portability
- ☛ multiway multiplexing, thread counter enhancements
- ☛ new platforms (Cray X-1 and AMD Opteron/K8)



# PAPI Perfometer



Jack Dongarra, Tennessee



# SvPablo

---

## ✍ **Graphical performance analysis environment**

- ✍ source code instrumentation
- ✍ performance data capture, browsing and analysis
- ✍ F77/F90 and C language support

## ✍ **Performance data capture features**

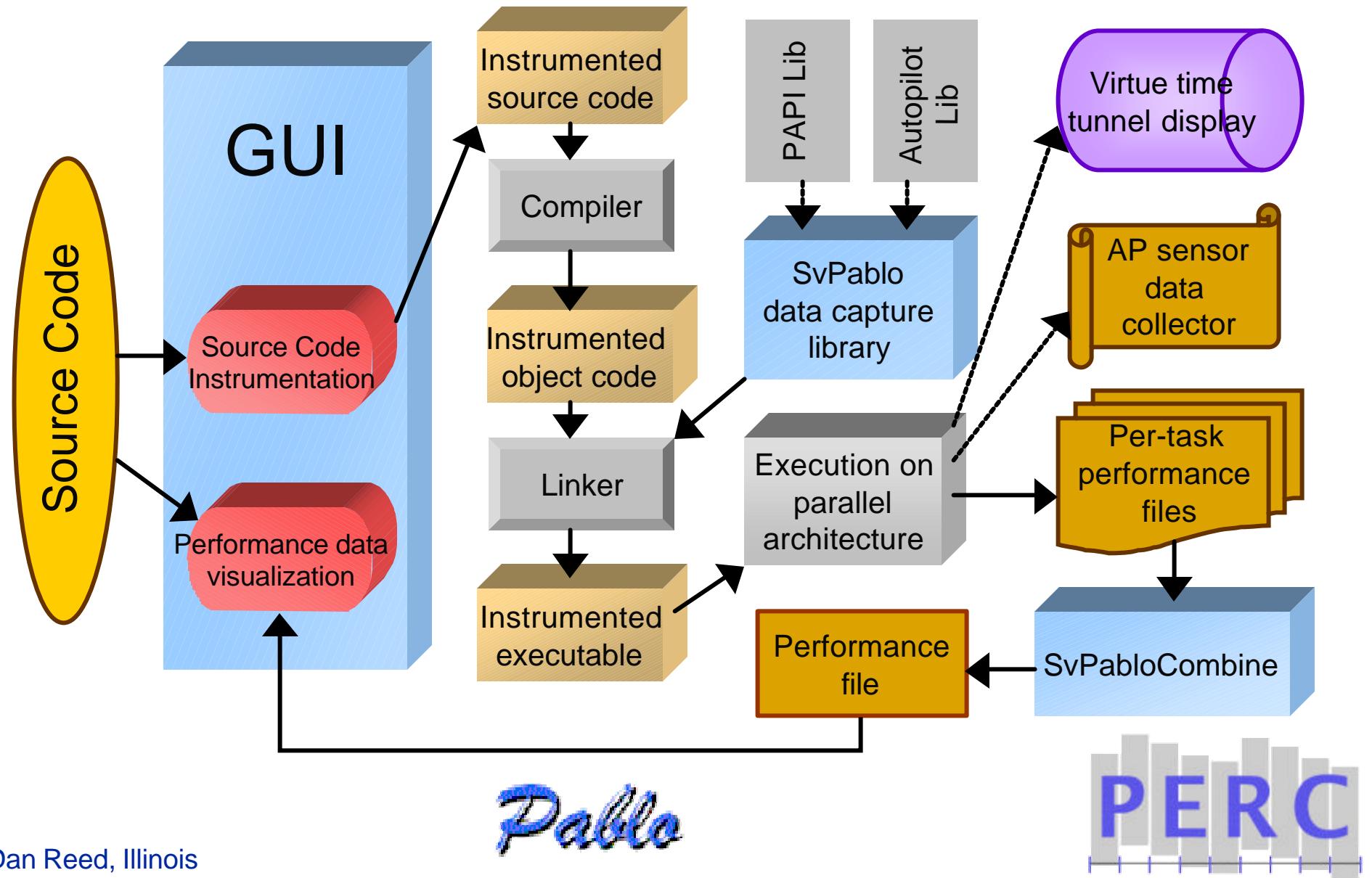
- ✍ software-based instrumentation (default)
- ✍ hardware performance counter data optional (via PAPI)
- ✍ statistical summaries for long-running codes (no traces)
- ✍ option for real-time data transmission via Autopilot

## ✍ **Platforms supported prior to SciDAC/PERC**

- ✍ Sun Solaris, IBM SP, SGI Origin, IA-32 Linux



# SvPablo Components



# SvPablo Enhancements via PERC

## ☞ Introduction of new features

- ☞ compact application signature models and C++ support (ROSE)
- ☞ creation of OpenMP support and dynamic overhead control (ongoing)
- ☞ support for scalability analysis (ongoing)

## ☞ Improved performance data capture

- ☞ integration to new PAPI releases
- ☞ MPI communication data capture
- ☞ control of instrumentation overhead
- ☞ dynamic instrumentation via Dyninst

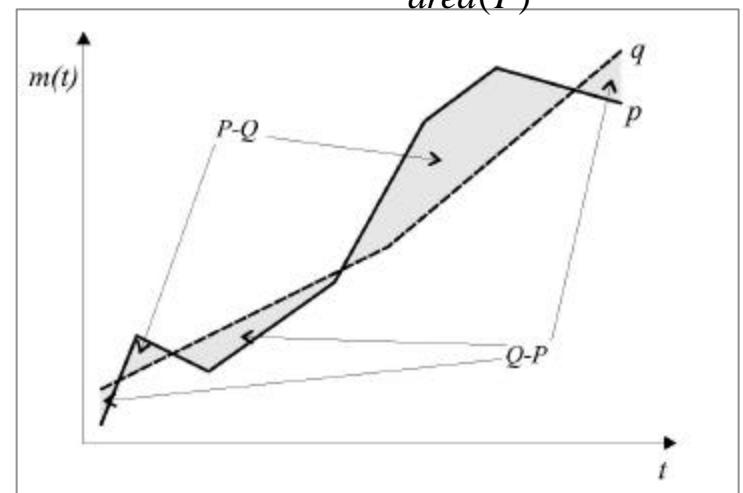
## ☞ Support for new platforms

- ☞ HP/Compaq Alpha and Linux IA-64
- ☞ NEC SX-6 and Sony PlayStation2

## ☞ Application characterizations and optimizations

- ☞ POP, PCTM, EVH1, MILC and AORSA-3D

$$DS(p, q) \leq 1 \leq \frac{\text{area}((P?Q)?(Q?P))}{\text{area}(P)}$$



# SvPablo POP Instrumentation

---

## ☞ POP: Parallel Climate Model/Ocean Simulation

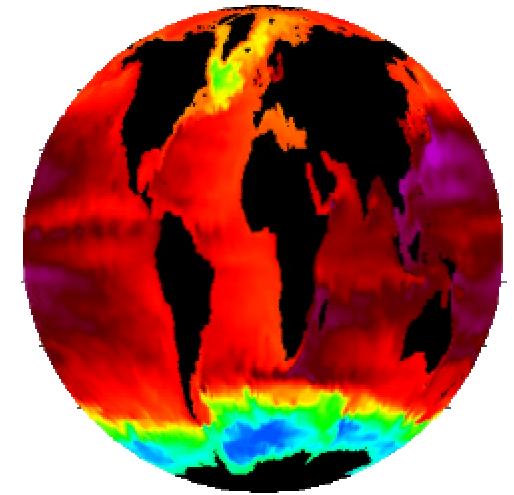
- ☞ all source code in F90 with MPI-based communication
- ☞ NERSC SP with PAPI support

## ☞ Original POP performance reporting

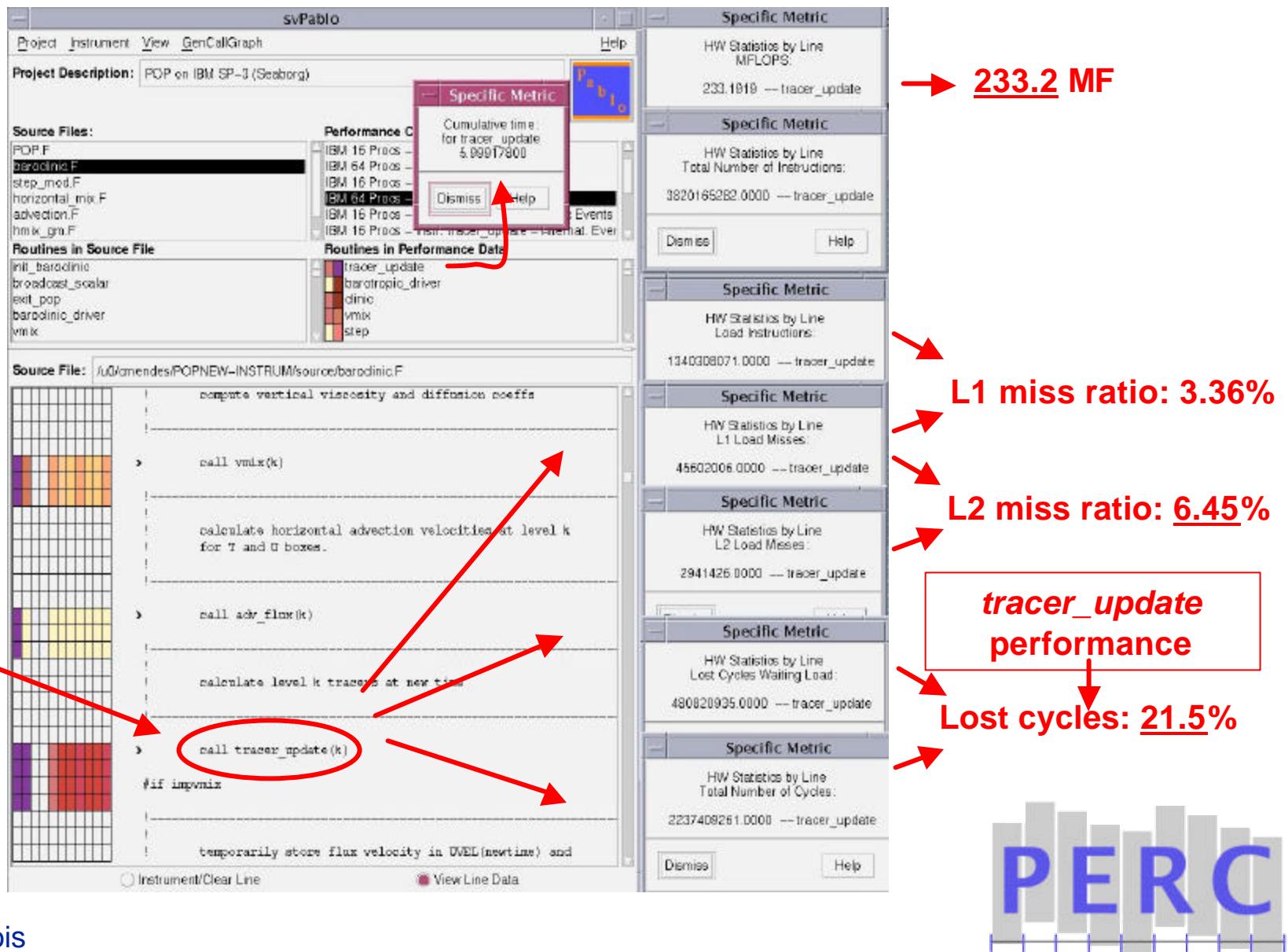
- ☞ good indication of critical code regions
- ☞ lack of quantitative performance data
- ☞ lack of indication of major performance factors

## ☞ SvPablo instrumentation strategy

- ☞ phase one: higher level routines
  - ☞ verify consistency of original POP reporting
- ☞ phase two: lower level critical routines
  - ☞ capture detailed performance data
  - ☞ understand causes of observed performance
  - ☞ propose changes for improving performance



# SvPablo POP Performance



# SvPablo POP: Lessons Learned

---

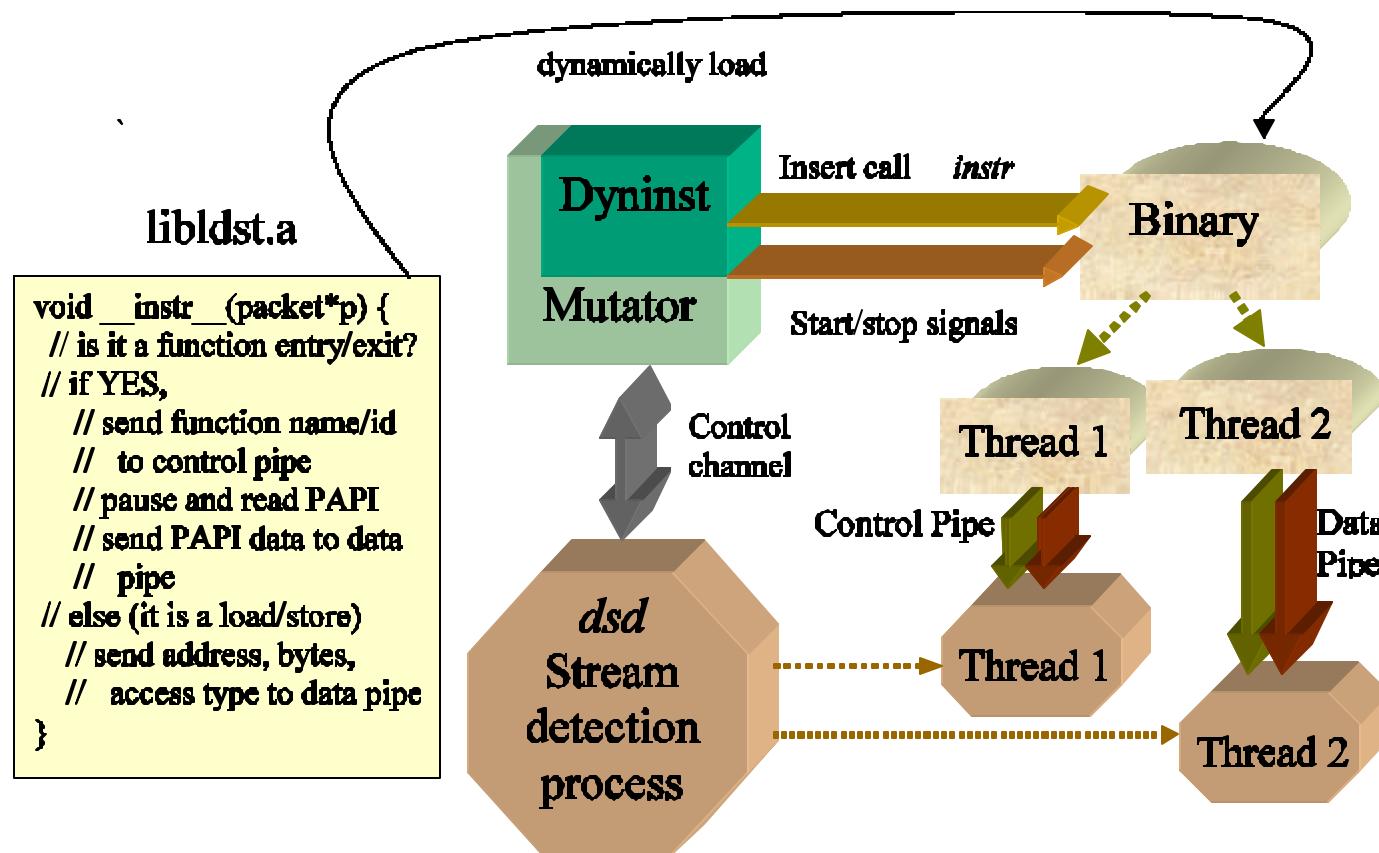
## ✍ Conclusions

- ✍ memory access is a key issue on POP-1.4.3
- ✍ high L2 miss-ratios imply more main memory traffic
- ✍ fat SMP nodes are not the best choice for POP-1.4.3

## ✍ Performance improvement guidelines

- ✍ minimize memory sharing between processors
- ✍ improve L2 miss ratio by code restructuring
- ✍ restructuring needs to be applied to major data structures
  - ✍ several code regions

# Dynamic Stream Detection Architecture



# dsd: Regularity Measurement Tool

---

## ✍ Instrumentation

- ✍ PAPI for bottlenecks
- ✍ Dyninst for transitory instrumentation
- ✍ user control of detection overhead
- ✍ binary instrumentation only (source code not needed)

## ✍ Applied to several codes

- ✍ Fortran and C
- ✍ SPEC, NAS benchmarks, and production codes

## ✍ Regularity data used to guide optimizations

- ✍ 4% improvement in gzip
  - ✍ inserting prefetch instructions
- ✍ 5% FT improvement
  - ✍ single function improved by 70%



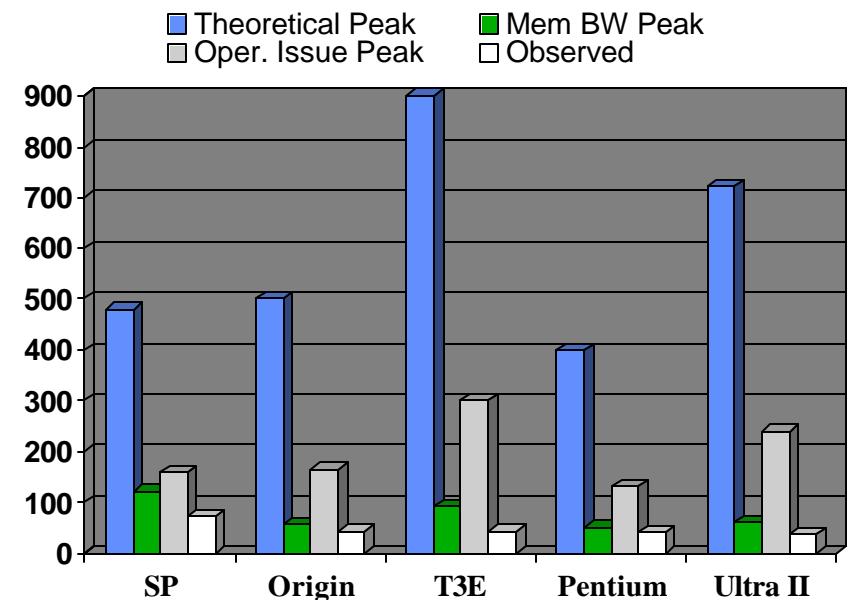
# Performance Bounding Tool

## ✍ Automate methodology

- ✍ Gropp, Kaushik, Keyes and Smith
- ✍ source code analysis to gather application signature
- ✍ coupled with machine signature to bound performance

## ✍ Rudimentary prototype for C/C++

- ✍ primitive memory analysis
- ✍ LLNL ROSE/Sage3 infrastructure
- ✍ machine signatures
- ✍ not yet coupled



# Annotation of 2-D Hall MHD (TOPS/CMRS)

```

// ...

/* Lap phi - U */
((f[j])[i]).phi = (((((x[j])[(i + 1)]).phi - two * ((x[j])[i]).phi) + ((x[j])[(i - 1)]).phi) * hydhw) * dhxdhy + (((((x[(j + 1)])[i]).phi - two * ((x[j])[i]).phi) + ((x[(j - 1)])[i]).phi) * hxdhy) * dhxdhy - ((x[j])[i]).U) * hxhy;
/* mem = 9; fop = 13 */

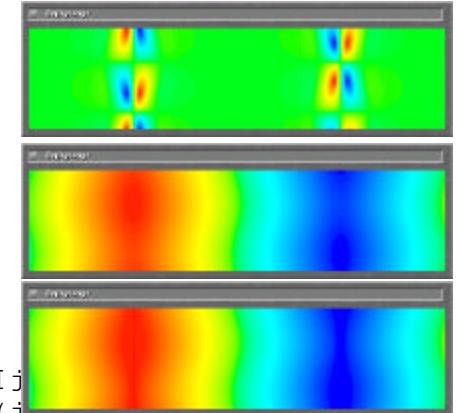
/* (1 - de^2 Lap) psi - F */
((f[j])[i]).psi = (((((x[j])[i]).psi - de2 * (((((x[j])[(i + 1)]).psi - two * ((x[j])[i]).psi) + ((x[j])[(i - 1)]).psi) * hydhw) * dhxdhy + (((((x[(j + 1)])[i]).psi - two * ((x[j])[i]).psi) + ((x[(j - 1)])[i]).psi) * hxdhy) * dhxdhy) - ((x[j])[i]).F) * hxhy; /* mem = 9; fop = 15 */

/* - nu Lap U + vx * U_x + vy * U_y - (B_x F_x + B_y F_y)/de2 */
((f[j])[i]).U = hxhy * ((-nu * (((((x[j])[(i + 1)]).U - two * ((x[j])[i]).U) + ((x[j] - 1)).U) * hydhw) * dhxdhy + (((((x[(j + 1)])[i]).U - two * ((x[j])[i]).U) + ((x[(j - 1)])[i]).U) * hxdhy) * dhxdhy) + (((vxp * (((((x[j])[i]).U - ((x[j])[i - 1]).U) * dhx) + vym * (((((x[j])[i]).U - ((x[j])[i - 1]).U) * dhy)) + vyp * (((((x[j])[i]).U - ((x[j])[i - 1]).U) * dhx + F_eq_x) + Bxm * (((((x[j])[i + 1]).U - ((x[j])[i]).U) * dhy) + Byp * (((((x[j])[i]).U - ((x[(j - 1)])[i]).U) * dhx + F_eq_x)) + Byp * (((((x[j])[i]).U - ((x[(j - 1)])[i]).U) * dhy)) + Bym * (((((x[(j + 1)])[i]).U - ((x[j])[i]).U) * dhy)) * dde2); /* mem = 23; fop = 49 */

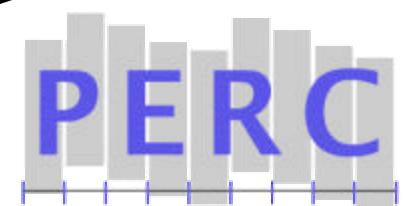
/* -nu Lap F + vx * F_x + vy * F_y - rho_s2 (B_x U_x + B_y U_y) */
((f[j])[i]).F = hxhy * ((-nu * (((((x[j])[(i + 1)]).F - two * ((x[j])[i]).F) + ((x[j] - 1)).F) * hydhw) * dhxdhy + (((((x[(j + 1)])[i]).F - two * ((x[j])[i]).F) + ((x[(j - 1)])[i]).F) * hxdhy) * dhxdhy) + (((vxp * (((((x[j])[i]).F - ((x[j])[i - 1]).F) * dhx + F_eq_x) + vym * (((((x[j])[i]).F - ((x[(j - 1)])[i]).F) * dhy)) + vyp * (((((x[j])[i]).F - ((x[(j - 1)])[i]).F) * dhx + F_eq_x)) + vym * (((((x[(j + 1)])[i]).F - ((x[j])[i]).F) * dhy)) + vyp * (((((x[j])[i]).F - ((x[(j - 1)])[i]).F) * dhx) + Bxm * (((((x[j])[i + 1]).U - ((x[j])[i]).U) * dhx)) + Byp * (((((x[j])[i]).U - ((x[(j - 1)])[i]).U) * dhy)) + Bym * (((((x[(j + 1)])[i]).U - ((x[j])[i]).U) * dhy)) * rhos2); /* mem = 23; fop = 49 */

} /* for loop totals: mem = 92 * (xinte - xints) + 1; fop = 170 * (xinte - xints) */
} /* for loop totals: mem = (92 * (xinte - xints) + 1) * (yinte - yints) + 1;
   fop = (170 * (xinte - xints)) * (yinte - yints) */

```



Implies 230 MF  
Bound on 2 GHz  
Xeon



# ROSE

## ✍ Simplify scientific software development

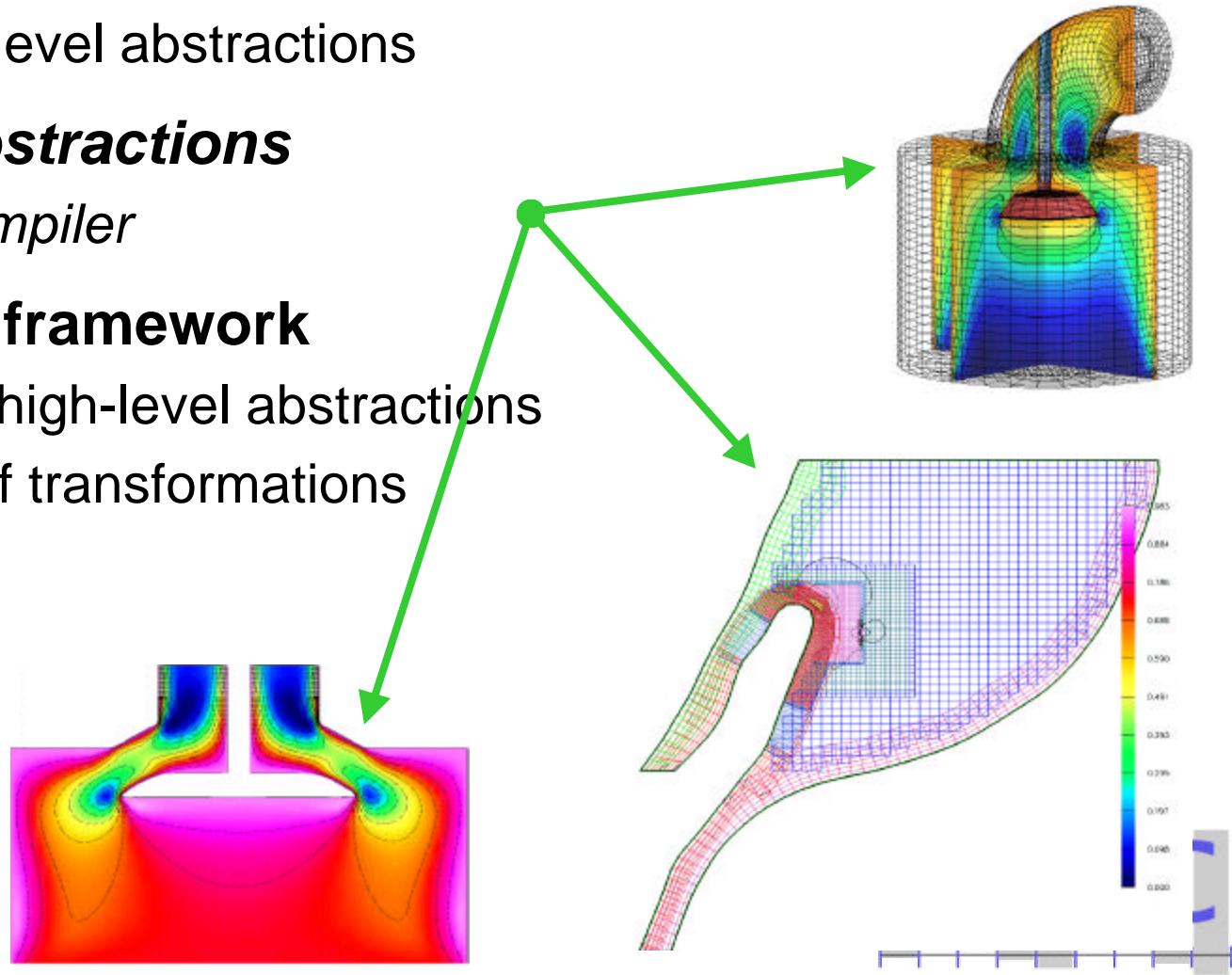
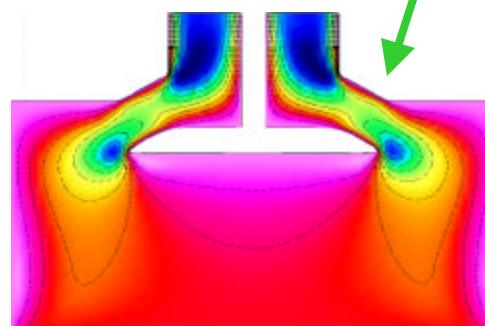
- ✍ use libraries and optimize the libraries at compile time
- ✍ optimize high-level abstractions

## ✍ User defined abstractions

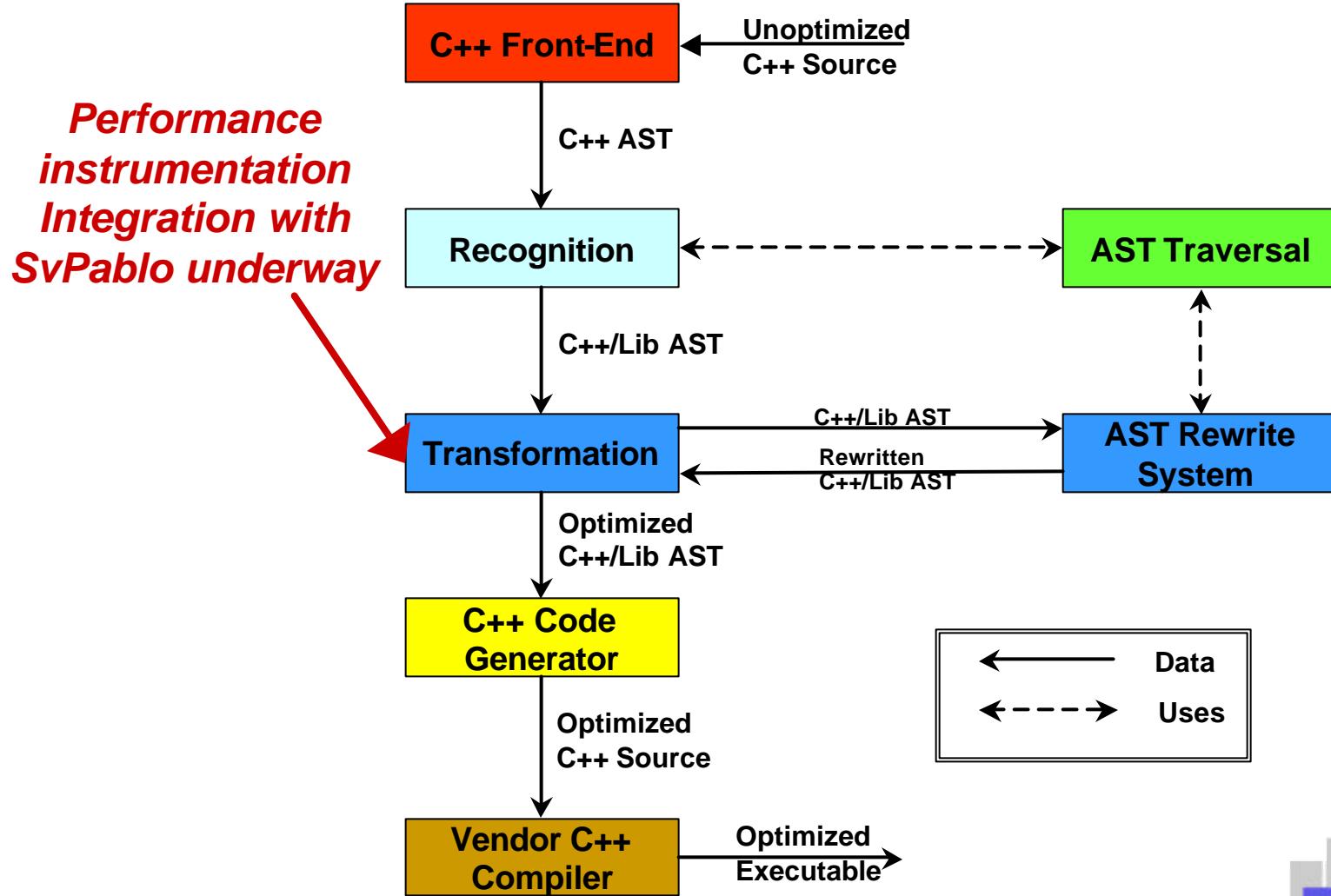
- ✍ ignored by compiler

## ✍ ROSE compiler framework

- ✍ recognition of high-level abstractions
- ✍ specification of transformations



# Overview of ROSE Approach



# Performance Assertions

---

## ✍ Traditional performance analysis

- ✍ focuses on data collection and management
- ✍ lacks semantic information about the rationale for the data

## ✍ New solution: Explicit performance assertions

- ✍ applications include expressions defining expectations
- ✍ similar to correctness assertions

## ✍ Benefits

- ✍ higher level of abstraction for performance analysis
- ✍ reduced data management by jettisoning raw data
- ✍ relative rather than absolute comparisons

## ✍ Prototype provides several compelling uses

- ✍ raising performance exceptions
- ✍ validating performance models
- ✍ enabling local performance-based adaptation



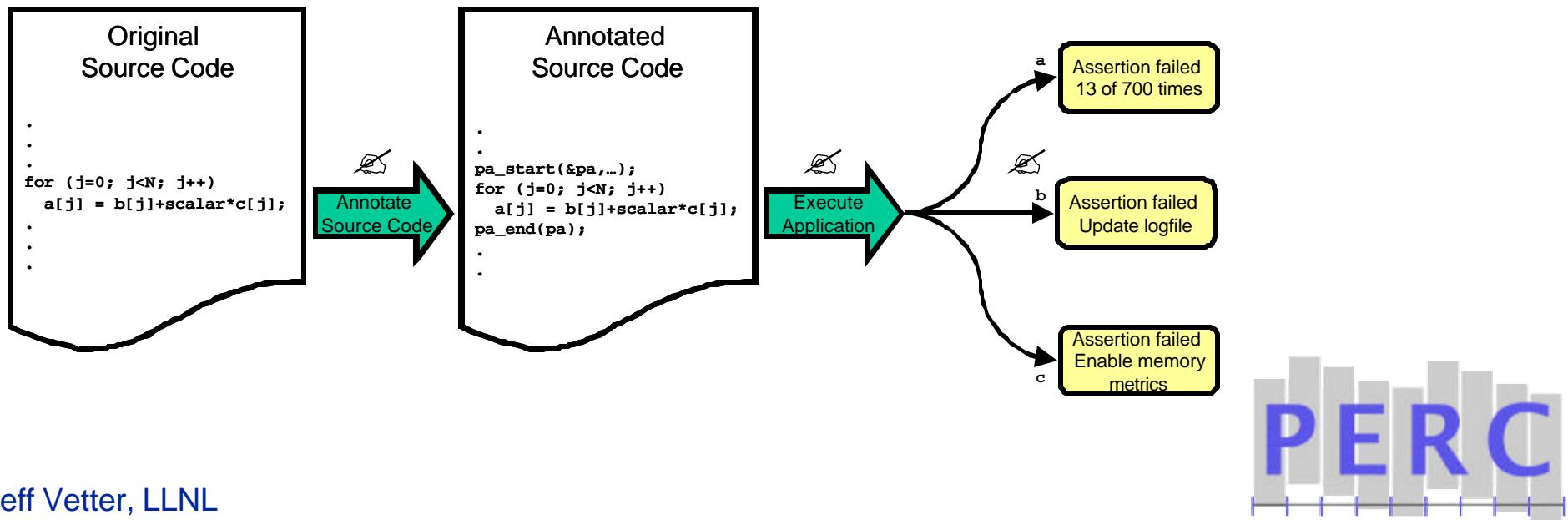
# Performance Assertion Overview

## ✍ Prototype design

- ✍ Performance Assertions (PA) library

## ✍ Process overview

- ✍ annotate source code with PA expressions
- ✍ execute application with PAs enabled
- ✍ review PA results of execution



# Example: Validating Performance Models

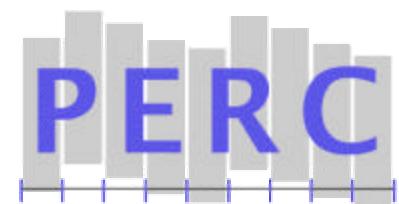
## ☞ Assertions to verify PETSc performance model explicitly

```
1:  pa_start(&pa, "$nFlops", PA_AEQ, "11 * %g * %g", &ym, &xm);
2:  for (j=ys; j<ys+ym; j++) {
3:    for (i=xs; i<xs+xm; i++) {
4:      if (i == 0 || j == 0 || i == Mx-1 || j == My-1) {
5:        f[j][i] = x[j][i];
6:      } else {
7:        u      = x[j][i];
8:        uxx   = (two*u - x[j][i-1] - x[j][i+1])*hydhx;
9:        uyy   = (two*u - x[j-1][i] - x[j+1][i])*hxdhy;
10:       f[j][i] = uxx + uyy - sc*PetscExpScalar(u);
11:     }
12:   }
13: }
14: pa_end(pa);
15: PetscLogFlops(11*ym*xm);
```

## ☞ Expression

- ☞ "\$nFlops", PA\_AEQ, "11 \* %g \* %g", &ym, &xm
- ☞ measure floating point operations with instrumentation
- ☞ test approximate equality ( $\pm 10\%$ ) to '11 \* ym \* xm' ?

## ☞ Empirical measurements verify the model



# PERC Successes

---

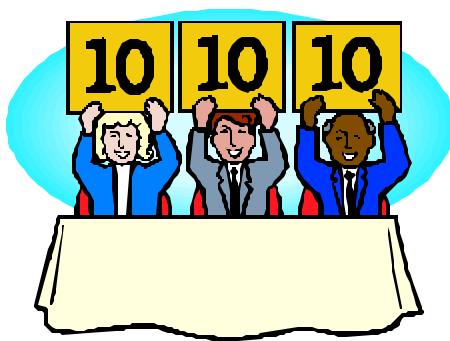
## ☞ PERC has brought together

- ☞ most major researchers in performance analysis
- ☞ a new set of ideas for measurement and modeling

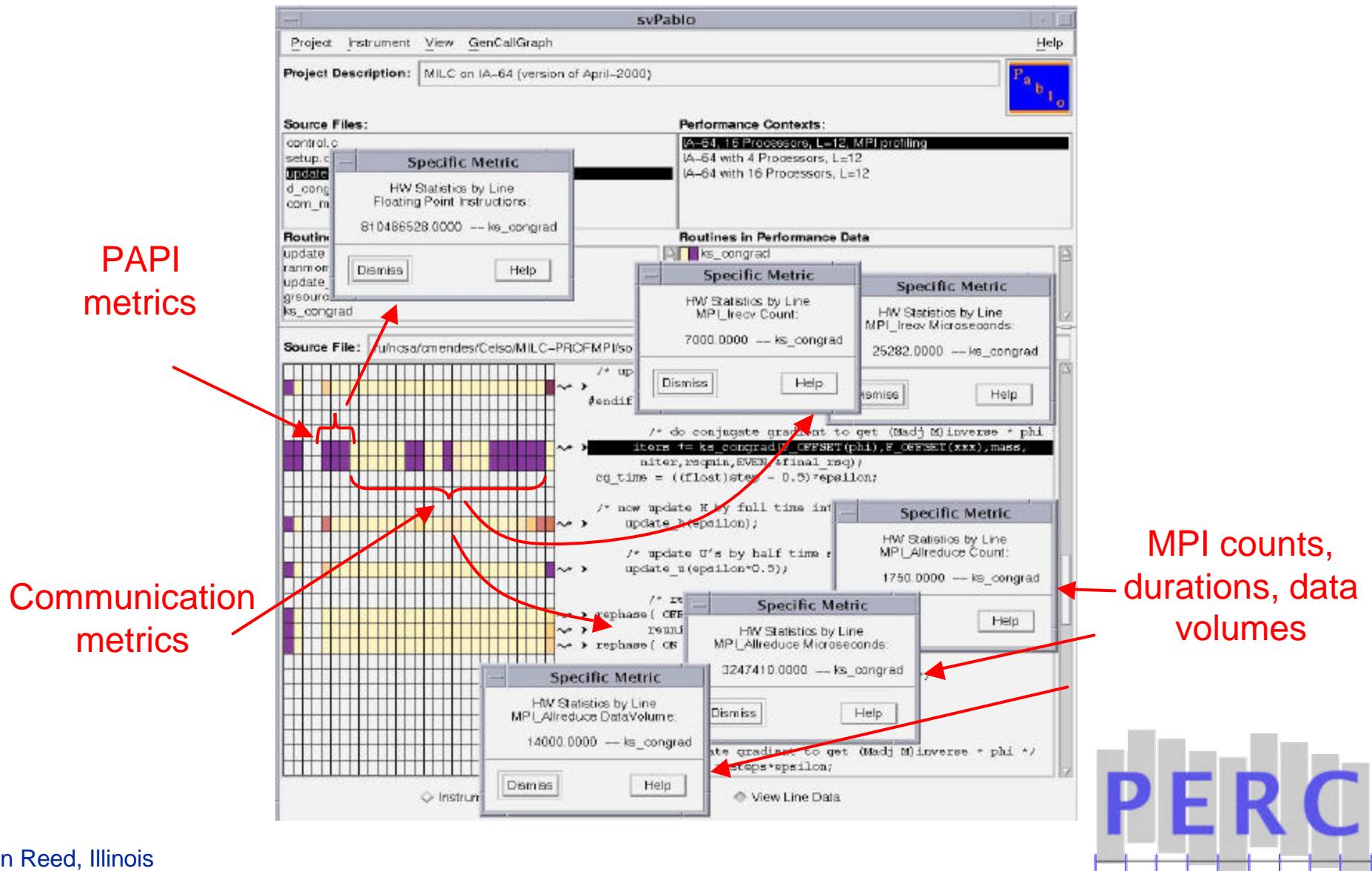
## ☞ It has also illuminated performance challenges

- ☞ critical SciDAC applications
- ☞ parallel architectures and system software

## ☞ We're making great progress!



# SvPablo Communication Performance Data



# SvPablo Ongoing Directions

---

## ✍ Future improvements

- ✍ C++ instrumentation and analysis
  - ✍ via SciDAC/PERC ROSE infrastructure (LLNL)
- ✍ Fortran parser replacement
  - ✍ more robust and extensible front-end
  - ✍ Open64 package is a possible alternative

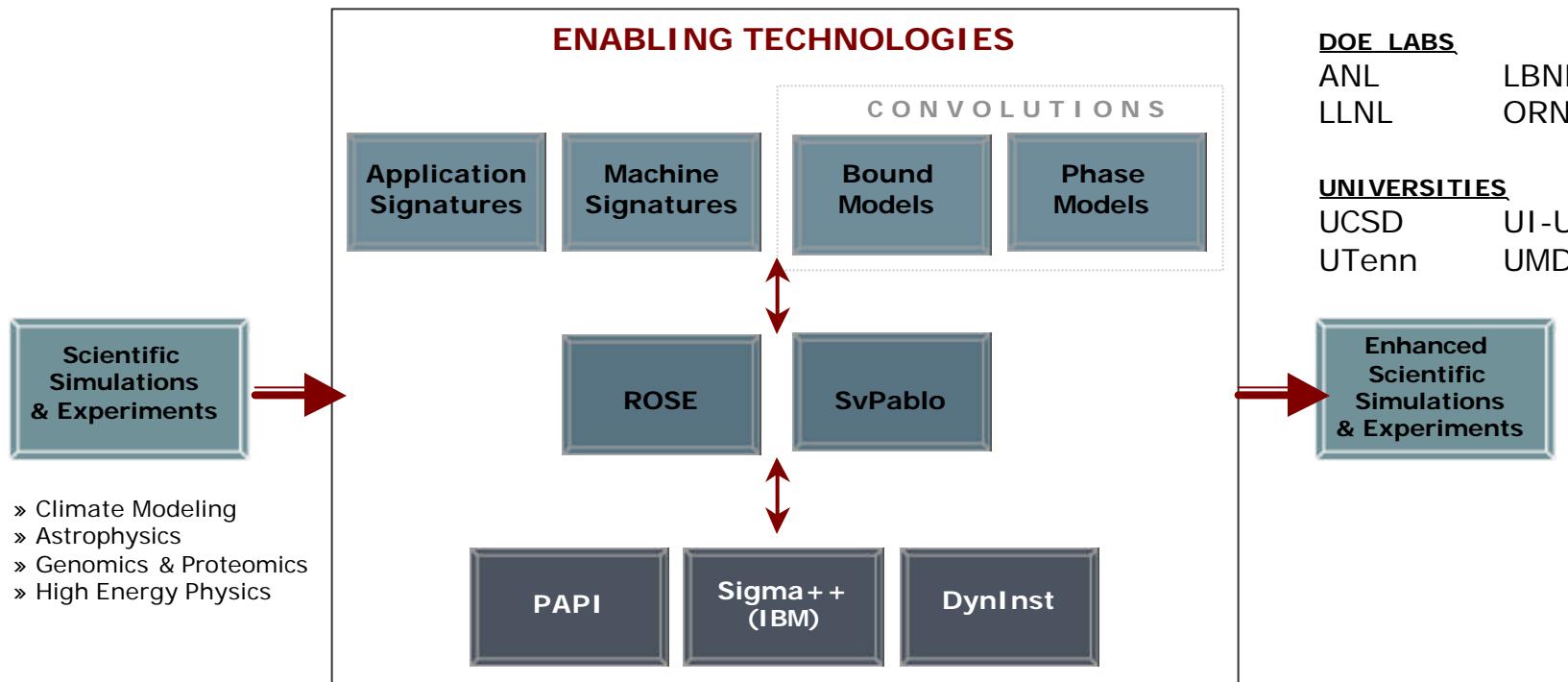
*Pablo*

## ✍ Currently under analysis/integration

- ✍ construction of application signature modeling
- ✍ exploration of distinct scalability scores
  - ✍ scalability analysis
- ✍ dynamic control of instrumentation overhead
  - ✍ via call graph analysis



**Developing a science for understanding performance of scientific applications on high-end computer systems, and engineering strategies for improving performance on these systems.**



#### GOALS

##### **Optimize and Simplify:**

- Profiling of real applications
- Measurement of machine capabilities  
(emphasis on memory hierarchy)
- Performance prediction
- Performance monitoring
- Informed tuning

Understand the key factors in applications that affect performance.  
 Understand the key factors in computer systems that affect performance.  
 Develop models that accurately predict performance of applications on systems.  
 Develop an enabling infrastructure of tools for performance monitoring, modeling and optimization.  
 Validate these ideas and infrastructure via close collaboration with DOE SC and other application owners.  
 Transfer the technology to end-users.

#### DOE LABS

ANL	LBNL
LLNL	ORNL

#### UNIVERSITIES

UCSD	UI-UC
UTenn	UMD

# Active Harmony

---

- ☞ **Real-time performance optimization**
  - ☞ automatic library selection (code)
    - ☞ monitor library performance and switch library if necessary
  - ☞ automatic performance tuning (parameters)
    - ☞ monitor system performance and adjust runtime parameters
  - ☞ minimal changes to application source code
    - ☞ applications adaptive to the external factors
- ☞ **Approach based on the Nelder-Mead simplex method**
  - ☞ modified to handle discrete value space
  - ☞ approximate the extreme of a function
    - ☞ considering the worst point of the simplex
    - ☞ forming its symmetrical image through the opposite (hyper) face
  - ☞ at each step
    - ☞ a better point replaces the worst point
    - ☞ this moves the simplex towards the extreme

# I/O Intensive Application Tuning

## 3-D volume reconstruction code

- built atop the Active Data Repository (ADR) middleware
- tunable parameters
  - memory tile size for data aggregation
  - pending reads

## Active Harmony performance improvement

- query processing speedup by up to 50%
  - 70 randomly generated queries
  - some of greatest speedups for short queries

